Imitation Learning CarRacingV2 DRL Project 1 Report

Name: Yisong Tang
Student ID: K12338084

In this project, I tackle the CarRacing environment using single-frame grayscale images (84×84) as input, rather than the typical 4-frame stack. Behavior Cloning achieved a mean score of approximately 550–600 (variance around 200). Then I applied the DAgger algorithm, which further improves the model, bringing its score close the expert model (around 700). My best recorded on the challenger server is about 694.946, very near the expert's performance.

1. Introduction

The CarRacing environment requires controlling a racing car on a procedurally generated track, avoiding off-track scenarios. In this project, we are asked to use a single 84×84 grayscale frame at each step. The expert policy I aim to mimic operates with 4-frame stacks and scores around 700, with a variance around 200.

Dataset Overview

- Train data: A set of .npz files, each containing a state of shape (84,84) and a discrete action (from 5 possible actions).
- Expert Model: Available to query expert actions using stacked 4-frame observations.
- Environment: CarRacing-v2 from Gym, resized to 84×84 in grayscale.

My goal is to build and refine a policy network that takes only one (84,84) frame as input, and apply Dagger algorithm to further improve the model.

2. Methodology

Neural Network Architecture:

I design a convolutional neural network for single-channel images. Its key layers are:

1. Convolutional Layers
    - Conv1: in_channels=1, out_channels=32, kernel_size=8, stride=4
    - Conv2: out_channels=64, kernel_size=4, stride=2
    - Conv3: out_channels=64, kernel_size=3, stride=1

   - Each followed by a ReLU activation

2. Fully Connected Layers
   - Flatten the final convolution output
   - FC1: 64*7*7 → 512, ReLU + Dropout(0.2)
   - FC2: 512 → 5 for discrete action logits

I apply Dropout and Weight Decay for regularization. BatchNorm is not applicable because of the input and LayerNorm can cause problem during uploading to the challenger server and transfer onnx to torch.

Behavior Cloning:

1. Data Loading
   I load (84,84) states and actions from .npz files in the train folder.
2. Data Processing
   Each (84,84) is expanded to (1,84,84) in the dataset's __getitem__, representing a single channel. Batching yields (batch_size, 1, 84, 84).
3. Training Loop
   - Optimizer: Adam with lr=1e-4 and weight_decay=1e-5
   - Loss: CrossEntropyLoss
   - I also compute policy entropy via Categorical(logits=logits).entropy() to observe the distribution's spread.
4. Validation
   A separate validation set monitors classification accuracy and loss.

In practice, pure BC yields a validation accuracy around 70–71%. Evaluating in the environment, it achieves an average score of 550–650 with a variance of ±200.

3. DAgger Approach

Algorithm Principle:

DAgger (Dataset Aggregation) exposes the policy to states from its own trajectory. Each step, I either use the expert's action (with probability β) or my current policy's action (with probability 1−β). Meanwhile, I always record the expert's action with the states for further training. This process gradually corrects potential errors on states that might otherwise never appear in the BC dataset.

Implementation and Configuration:

- max_steps=4000 per DAgger iteration
- β starts at 1.0 and decays to 0.9, then 0.81, etc.
- All new data is stored in a dagger_train/ folder (train data also copied into it), keeping track of both old and new samples.
- Same Adam optimizer with lr=1e-4, weight_decay=1e-5

Each iteration proceeds as follows:

1. Interact with the environment using a β-mixture of expert and student actions.
2. Log “single-frame + expert action” pairs.
3. Append them to the dataset.
4. Retrain for a few epochs on the expanded dataset.

4. Results and Analysis

1. Expert Policy: Averages 600–700 over 10 runs, with variance ~200. Scores sometimes exceed 900, sometimes hit low around 300-400.
2. Pure BC: Achieves around 550–650 on average.
3. DAgger: After several iterations, the policy consistently approaches ~670 average score. My best test result reached 694.946, which is extremely close to the expert’s performance. Due to the environment’s stochastic nature, consistently surpassing 700 may require additional training or favorable tracks.

5. Conclusion, Improvements and Reflection

Conclusion:

- A single-frame grayscale policy can nearly match expert performance in CarRacing through demonstration-based learning.
- Simple BC yields ~550–650, and DAgger further refines it to nearly 700 on average. If with better expert model, with Dagger, the model can reach higher score.
- Variance is still large because tracks are randomly generated, so I typically run

multiple episodes to get a stable mean.

Possible Improvements:

- Data Augmentation: Random cropping, noise, or other techniques could improve robustness.
- More Advanced Architectures: Incorporate self-attention or normalization layers (if ONNX constraints can be handled).
- Longer DAgger Iterations: Gather more expert corrections in tricky situations, especially during high-speed turns.

Personal Reflection:

Working on this project has highlighted that even with limited visual context (one frame), it is possible to learn a strong CarRacing policy, especially when leveraging expert demonstrations and iterative feedback through DAgger. The randomness of CarRacing remains a key factor in final score variance.